THE BULLETIN OF SYMBOLIC LOGIC
Volume 25, Number 3, September 2019

ETA-RULES IN MARTIN-LÖF TYPE THEORY

ANSTEN KLEV

Abstract. The eta rule for a set A says that an arbitrary element of A is judgementally identical to an element of constructor form. Eta rules are not part of what may be called canonical Martin-Löf type theory. They are, however, justified by the meaning explanations, and a higher order eta rule is part of that type theory. The main aim of this article is to clarify this somewhat puzzling situation. It will be argued that lower order eta rules do not, whereas the higher order eta rule does, accord with the understanding of judgemental identity as definitional identity. A subsidiary aim is to clarify precisely what an eta rule is. This will involve showing how such rules relate to various other notions of type theory, proof theory, and category theory.

§1. Introduction.

The rules governing the constants of Martin-Löf type theory are usually grouped into four classes: formation rules, introduction rules, elimination rules, and equality rules. Sometimes also a fifth class is considered, containing for instance the following two identities:

$$\lambda([x]\mathsf{ap}(c,x)) = c : \Pi(A,B),$$
$$\mathsf{pair}(\mathsf{fst}(c),\mathsf{snd}(c)) = c : \Sigma(A,B).$$

The rules of this class may be called eta rules owing to their similarity to the rule of η-reduction in the lambda calculus,

$$\lambda x.fx \twoheadrightarrow f.$$

For reasons that will soon become clear, I shall also call rules of this fifth kind *lower order* eta rules. The interpretation of typed lambda calculus in cartesian closed categories suggests the name 'uniqueness principle,' a term employed in the HoTT-book [34] for these rules.

The status of the four first classes of rules is quite unproblematic. Formation rules introduce sets, that is, types of individuals. Introduction rules introduce canonical elements of sets and serve to justify formation rules. Elimination rules are needed whenever one wishes (as one often does) to define a function by induction on a set. Equality rules serve to justify the elimination rules.

The status of eta rules is less clear. They are not part of what may be called canonical Martin-Löf type theory, namely, the version of type theory presented in, for instance, [23]. Nevertheless, they are justified by the

Received May 7, 2018.
2010 *Mathematics Subject Classification.* 03B15, 03A05.
Key words and phrases. type theory, definitional identity, justification of logical laws.

so-called meaning explanations. Moreover, the higher order eta rule

$$[x]f(x) = f : (x : \alpha)\beta$$

pertaining to the 'logical framework,' is part of canonical type theory. Lower order eta rules are appealed to at a few places in the HoTT-book. But the book seems to leave it up to its readers to decide whether or not to accept these rules in general, since they are there described as 'optional.'

There are therefore several reasons for being interested in eta rules. Readers of the HoTT-book may want to know what is at stake when deciding whether to accept or reject such rules. Those interested in the foundations of Martin-Löf type theory may want to understand why eta rules are not accepted in the canonical version of the theory, although they are justified by the meaning explanations. Moreover, they may want to understand why the higher order eta rule is accepted, in particular why the reasons for not accepting the lower order eta rules do not apply to the higher order rule.

In the latter part of this article I shall argue that lower order eta rules do not, whereas the higher order eta rule does, accord with the understanding of judgemental identity—identity as expressed by judgements of the form $a = b : \alpha$ and $\alpha = \beta : type$—as definitional identity. If one wishes to understand judgemental identity as definitional identity, one should therefore not accept lower order eta rules as axioms.

Reflection on eta rules thus forces us to deal with certain fundamental aspects of type theory, in particular the proper understanding of judgemental identity and the role of the meaning explanations. Although eta rules are justified by the meaning explanations, there are nevertheless reasons not to accept them. The meaning explanations should thus not be regarded as the final arbiter in the question of which rules to accept.

Judgemental identity is sometimes equated, without further comment, with definitional identity. But, clearly, the mere form of an identity judgement does not make it into a judgement of definitional identity: to merit that title the judgement must conform to our preferred account of definitional identity.

Before getting into all of this (Sections 9–12), I shall offer a systematic overview of eta rules and certain closely related notions. In particular, I shall distinguish eta rules from co-eta rules. Whereas an eta rule says that any element of a set is judgementally identical to an element of constructor form, a co-eta rule says that any function defined on a set A applied to an arbitrary element of A is judgementally identical to an element of selector form (Sections 3 and 7). I shall also briefly discuss the incarnations of eta rules in category theory (Section 6) and proof theory (Section 8), as well as the propositional versions of eta rules (Section 5). Sections 6 and 8 are not essential to the argument of the article, but they provide additional context. I begin (Section 2) with certain preliminaries on the version of type theory that will be assumed.

§2. The hierarchy of higher types. I will assume the higher order formulation of Martin-Löf type theory, presented, for instance, in [23, Chapters 19–20] and [24]. This formulation is characterized by the presence of a hierarchy of higher types, or a 'logical framework' [12], which allows for the typing of all the operators introduced by the formation, introduction, and elimination rules, thus for instance of Π, Σ, λ, and ap. In the more well-known lower order formulation of the theory, found for instance in the classical references [20–22], these operations are not typed. We can say in the metalanguage there that, for instance, Π is a function taking a set A and a family of sets B over A and yielding a set $(\Pi x : A)B$; but there is no type of such functions in the system. A symbol such as Π therefore never occurs isolated in the lower order formulation, but always in a composition of the form $(\Pi x : A)B$. In traditional grammatico-logical terminology, Π may be called a *syncategorematic* term: it has no meaning in isolation, but only in composition with other terms. In the higher order formulation, by contrast, Π and all of the other symbols introduced by the formation, introduction, and elimination rules are *categorematic* terms, since each such symbol there signifies an object of some type in the hierarchy of higher types.

At the bottom of the hierarchy there is a type *set* and for each $A : set$, a type $el(A)$ of the elements of A. We shall follow the standard practice of writing A instead of $el(A)$. It is important to appreciate that the type *set* is not a universe in the usual sense of type theory [22, pp. 87–91]. A universe $\mathcal{U}$ is itself a set, hence, according to the meaning explanations, it has been laid down how the canonical elements of $\mathcal{U}$ are formed and how equal canonical elements of $\mathcal{U}$ are formed. This is done by means of the $\mathcal{U}$-introduction rules, which mirror a specified collection of formation rules. The universe $\mathcal{U}$ is therefore inductively defined and (provided it has only finitely many introduction rules) allows for the formulation of an elimination rule, encapsulating a principle of proof by induction [23, Chapter 14]. The type *set*, by contrast, is not inductively defined and does not support proof by induction. It is open and allows for the addition of new inhabitants A provided one specifies what the canonical elements of A are and what equal canonical elements of A are.

Some terminology: an introduction rule will sometimes be spoken of as associated with a set A, rather than with a set former Φ; likewise for formation, elimination, and equality rules, as well as eta rules. The constants introduced by the introduction rules for a set A are called its constructors and the constant introduced by the elimination rule for A is called its selector. For instance, the constructors of the set $\mathbf{N}$ of natural numbers are 0 and the successor function s; and its selector is the recursor R. If the n-ary function con is a constructor of the set A, then an element of constructor form $\mathsf{con}(a_1, \ldots, a_n)$ is also called a canonical element of A, or an element of canonical form. I shall speak of judgemental, definitional, and propositional *identity* and avoid the word 'equality' except in the case of the well-established term 'equality rule.' It seems to me preferable to think of the notion of equality as captured rather by equivalence relations on sets.

With some experience one can see how to 'derive' the elimination and equality rules for A from its introduction rule(s) [1]. Not all rules with a conclusion of the form $a : A$ can serve as an introduction rule for A. In particular, A may not appear negatively in the premiss of its own introduction rule (it may well appear positively, as in **N**-introduction). All of this is captured in the general scheme of rules formulated by Dybjer [6, 7]. This general scheme is useful not only for understanding the structure of the rules of type theory: it is also useful for metamathematical purposes, since the rules are thereby specified once and for all, whence the openness of the type *set* is somehow regimented.

The distinction between *set* and a universe $\mathcal{U}$ is an instance of the more general distinction between types and sets. Any set A gives rise to a type $el(A)$, but not every type is, or gives rise to, a set. The type *set*, for instance, is not itself a set. Types of the form $el(A)$ are types of individuals. A function is not an individual: it is an object of higher type. Function types are generated by the following rule:

$$\frac{\alpha : type \qquad x : \alpha \vdash \beta : type}{(x : \alpha)\beta : type.}$$

The identity of function types is governed by the rule

$$\frac{\alpha = \alpha' : type \qquad x : \alpha \vdash \beta = \beta' : type}{(x : \alpha)\beta = (x : \alpha')\beta' : type.} \qquad (\xi\text{-}type)$$

When β does not depend on α, the type $(x : \alpha)\beta$ is usually written $(\alpha)\beta$.

With a function $f : (x : \alpha)\beta$ there is associated a primitive notion of application, written $f(a)$ for arbitrary $a : \alpha$. This is captured by the following rule:

$$\frac{f : (x : \alpha)\beta \qquad a : \alpha}{f(a) : \beta[a/x].} \qquad \text{(App)}$$

Identical functions applied to identical arguments yield identical results:

$$\frac{f = g : (x : \alpha)\beta \qquad a = b : \alpha}{f(a) = g(b) : \beta[a/x].} \qquad \text{(App-cong)}$$

We usually write $f(a_1, \dots, a_n)$ for $f(a_1)\dots(a_n)$. It should be emphasized that whereas the application of a function $f : (x : \alpha)\beta$ to an argument $a : \alpha$ is a primitive notion, an element $c : \Pi(A, B)$ is applied to an $a : A$ by means of the application function ap, yielding $\mathsf{ap}(c, a)$. An element $c : \Pi(A, B)$ is an individual and therefore not a function in the proper sense.

An object of type $(x : \alpha)\beta$ can be formed by means of abstraction:

$$\frac{x : \alpha \vdash b : \beta}{[x]b : (x : \alpha)\beta.} \qquad \text{(Abs)}$$

We shall sometimes indicate the type of x in $[x]b$ by writing $[x : \alpha]b$.

The rule (Abs) should not be regarded as an introduction rule for the type $(x : \alpha)\beta$. Functions of type $(x : \alpha)\beta$ are also introduced through the formation, introduction, and elimination rules associated with sets. For instance, the rule of Π-formation

$$\Pi : (X : set)((X)set)set \qquad (\Pi\text{-form})$$

introduces a function, Π, that is not of the form $[x]b$. Owing to the openness of the type *set*, the range of functions that may be introduced in this way is also open. Hence one cannot hope to specify the canonical objects of type $(x : \alpha)\beta$ as being either of the form $[x]b$ or else taken from a list of constants, since such a list of constants cannot be specified once and for all. Function types are explained, not by the introduction-like rule (Abs), but rather by the elimination-like rules (App) and (App-cong). (Here 'cong' is short for 'congruence.')

A function of the form $[x]b$ is defined by the beta rule:

$$\frac{x : \alpha \vdash b : \beta \qquad a : \alpha}{([x]b)(a) = b[a/x] : \beta[a/x].} \tag{β}$$

Functions f and g are judgementally identical if they are pointwise judgementally identical:

$$\frac{f, g : (x : \alpha)\beta \qquad x : \alpha \vdash f(x) = g(x) : \beta}{f = g : (x : \alpha)\beta.} \tag{Ext}$$

The variable x here is to serve as an arbitrary argument to f and g, hence it may not occur free in any of them. Using (Ext) and (β) one can easily prove that judgementally identical terms abstract to judgementally identical functions, the so-called ξ-rule:

$$\frac{x : \alpha \vdash b = b' : \beta}{[x]b = [x]b' : (x : \alpha)\beta.} \tag{ξ}$$

One can also easily prove the higher order eta rule:

$$[x]f(x) = f : (x : \alpha)\beta. \tag{η}$$

The abstraction in $[x]f(x)$ is not to affect f, hence x may not occur free in f. The rule (η) is thus a derived rule of the system. It can also be taken as an axiom together with (ξ), since from these one can derive (Ext).

In the lower order formulation of type theory, operators such as Π, Σ, λ, as well as the selectors E, D, and J associated with Σ, $+$, and **Id**, respectively, are all variable-binding. In the higher order formulation, abstraction $[x]b$ and function-type formation $(x : \alpha)\beta$ are the only variable-binding operations. So, for instance, if $A : set$, $x : A \vdash B : set$, and $x : A \vdash b : B$, then in the higher order formulation, we write $\Pi(A, [x]B)$ rather than $(\Pi x : A)B$, and $\lambda([x]b)$ rather than $\lambda x.b$. Here λ is a function typed as follows:

$$\lambda : (X : set)(Y : (X)set)((x : X)Y(x))\Pi(X, Y). \tag{Π-intro}$$

One sees from the typing of λ that it is in fact a ternary function, whose first two arguments are a set A and a set-valued function $B : (A)set$. When all the arguments are written out we speak of a monomorphic notation. The notation employed for instance in $\lambda(f)$, by contrast, is called polymorphic, since here λ appears as a 'typically ambiguous' function. We shall mostly prefer the polymorphic notation, but it serves us merely as shorthand for the official, monomorphic notation.

Formation, introduction, and elimination rules may, in the higher order formulation, be written as type declarations. We have already seen two examples in (Π-form) and (Π-intro). We may, however, also employ the usual inference-rule style:

$$\frac{c : \Pi(A,B) \qquad C : (\Pi(A,B))set \qquad d : (z : (x : A)B(x))\, C(\lambda(z))}{\mathsf{F}(A,B,C,c,d) : C(c).} \tag{Π-elim}$$

An equality rule is always written as an identity:

$$\mathsf{F}(A,B,C,\lambda(f),d) = d(f) : C(\lambda(f)). \tag{Π-eq}$$

The rule (Π-elim) allows the definition of a function on $\Pi(A,B)$ by induction: to define a function from $\Pi(A,B)$ into some family $C : (\Pi(A,B))set$ it is enough to determine the value of the function for canonical terms $\lambda(f)$ of $\Pi(A,B)$.

From the selector F one can define the application function ap and derive the beta rule for Π-sets:

$$\mathsf{ap}(\lambda(f),a) = f(a) : B(a). \tag{Π-β}$$

§3. Formulation of eta rules. The beta rule for Π-sets suggests the formulation of an eta rule for such sets:

$$\lambda([x]\mathsf{ap}(c,x)) = c : \Pi(A,B). \tag{Π-η}$$

The variable x may not occur free in c. Read from right to left, (Π-η) says that the arbitrary element c of $\Pi(A,B)$ is judgementally identical to a term of constructor form. This reading of (Π-η) suggests that eta rules can be formulated also for other sets. Indeed, we should be able to formulate an eta rule for any set former that has just one associated introduction rule.

One such set former is Σ. For it, an eta rule may be formulated as follows:

$$\mathsf{pair}(\mathsf{fst}(c),\mathsf{snd}(c)) = c : \Sigma(A,B). \tag{Σ-η}$$

Here pair is the constructor for Σ-sets, and fst and snd are the two projections, definable from the selector for Σ.

From (Π-η) and (Σ-η) one can discern a general pattern. Let $A : set$ have just one constructor, the n-ary function con. Assume that we can define n functions $\mathsf{con\text{-}proj}_k$ on A satisfying

$$\mathsf{con\text{-}proj}_k(\mathsf{con}(a_1,\ldots,a_n)) = a_k : \alpha$$

for suitable types α. The functions $\mathsf{con\text{-}proj}_k$ may be regarded as generalized projection functions or generalized left inverses to con.[1] Then the eta rule for A is

$$\mathsf{con}(\mathsf{con\text{-}proj}_1(c),\ldots,\mathsf{con\text{-}proj}_n(c)) = c : A.$$

This identity says in effect that the projection functions $\mathsf{con\text{-}proj}_k$ together constitute a right inverse to con.

Let us see how this general pattern applies in the case of Π. The constructor λ is a ternary function, but, for the time being, let us treat it as it appears in

[1]Landin's original use of the term 'selector' [17, p. 310] appears to be precisely for such generalized projection functions.

the polymorphic notation, namely, as a unary function. For arbitrary $x : A$ we have

$$\mathsf{ap}(\lambda(f), x) = f(x) : B(x),$$

by (Π-β). Hence by (ξ) and (η) we get

$$[x]\mathsf{ap}(\lambda(f), x) = [x]f(x) = f : (x : A)B(x).$$

Let the projection function λ-proj on $\Pi(A, B)$ be defined as

$$\lambda\text{-}\mathsf{proj} = [z][x]\mathsf{ap}(z, x) : (\Pi(A, B))(x : A)B(x).$$

Then

$$\lambda\text{-}\mathsf{proj}(\lambda(f)) = [x]\mathsf{ap}(\lambda(f), x) = f : (x : A)B(x),$$

thus λ-proj is a left inverse to λ. Moreover,

$$\lambda(\lambda\text{-}\mathsf{proj}(c)) = \lambda([x]\mathsf{ap}(c, x)) : \Pi(A, B),$$

so we may indeed formulate (Π-η) as

$$\lambda(\lambda\text{-}\mathsf{proj}(c)) = c : \Pi(A, B).$$

Now let us regard λ as it in fact is, namely, as a ternary function whose closure has the form $\lambda(A, B, f)$, where $A : set$, $B : (A)set$, and $f : (x : A)B(x)$. The general formulation of eta rules requires that we define three projection functions. It is clear that $\mathsf{con\text{-}proj}_k$, written monomorphically, is not in general a unary function. For instance, it is clear that any of the three projection functions λ-proj_k, to be defined on $\Pi(A, B)$, should take A as its first argument and B as its second argument. Thus, we want:

$$\begin{aligned}
&\lambda\text{-}\mathsf{proj}_1(A, B, \lambda(A, B, f)) = A : set,\\
&\lambda\text{-}\mathsf{proj}_2(A, B, \lambda(A, B, f)) = B : (A)set,\\
&\lambda\text{-}\mathsf{proj}_3(A, B, \lambda(A, B, f)) = f : (x : A)B(x).
\end{aligned}$$

But the functions λ-proj_1 and λ-proj_2 are easily defined, and λ-proj_3 is just the monomorphic version of the function λ-proj defined above.

Let us recall the well-ordering set $\mathbf{W}(A, B)$, where $A : set$ and $B : (A)set$. It has one constructor, given by the introduction rule

$$\frac{a : A \qquad b : (B(a))\mathbf{W}(A, B)}{\mathsf{sup}(a, b) : \mathbf{W}(A, B).} \qquad (\mathbf{W}\text{-intro})$$

Any $\mathsf{sup}(a, b) : \mathbf{W}(A, B)$ may be thought of as a well-founded tree obtained by adding a root connecting the well-founded trees enumerated by $b : (B(a))\mathbf{W}(A, B)$. From the selector for $\mathbf{W}$ we may define the required projection functions

$$\begin{aligned}
&\mathsf{sup\text{-}proj}_1 : (\mathbf{W}(A, B))A,\\
&\mathsf{sup\text{-}proj}_2 : (z : \mathbf{W}(A, B))\big(B(\mathsf{sup\text{-}proj}_1(z))\big)\mathbf{W}(A, B)
\end{aligned}$$

satisfying the following identities:

$$\begin{aligned}
&\mathsf{sup\text{-}proj}_1(\mathsf{sup}(a, b)) = a : A,\\
&\mathsf{sup\text{-}proj}_2(\mathsf{sup}(a, b)) = b : (B(a))\mathbf{W}(A, B).
\end{aligned}$$

Thus we may formulate the eta rule for $\mathbf{W}$ as follows:

$$\mathsf{sup}(\mathsf{sup\text{-}proj}_1(c), \mathsf{sup\text{-}proj}_2(c)) = c : \mathbf{W}(A, B). \qquad (\mathbf{W}\text{-}\eta)$$

The unit type **1** has one constructor, namely, the constant

$$0_{\mathbf{1}} : \mathbf{1}. \qquad (\mathbf{1}\text{-intro})$$

This constructor is not a function, hence no projection functions can be defined in this case. The eta rule for **1** is therefore degenerate:

$$0_{\mathbf{1}} = c : \mathbf{1}. \qquad (\mathbf{1}\text{-}\eta)$$

The result of imposing this rule on **1** has been called the extensional unit type by Hofmann [14].

Let us also consider propositional identity, $\mathbf{Id}(A, a, a)$ for $A : \mathit{set}$ and $a : A$. Its constructor is

$$\mathsf{refl} : (X : \mathit{set})(x : X)\mathbf{Id}(X, x, x). \qquad (\mathbf{Id}\text{-intro})$$

Thus for any $A : \mathit{set}$ and $a : A$, we have a canonical element $\mathsf{refl}(A, a) : \mathbf{Id}(A, a, a)$. In this case it is essential to write the required projection functions in the monomorphic notation. Since the constructor refl is binary, we want two projection functions $\mathsf{refl\text{-}proj}_k$ satisfying

$$\mathsf{refl\text{-}proj}_1(A, a, \mathsf{refl}(A, a)) = A : \mathit{set},$$
$$\mathsf{refl\text{-}proj}_2(A, a, \mathsf{refl}(A, a)) = a : A.$$

But these are readily defined. The general scheme for eta rules yields

$$\mathsf{refl}(\mathsf{refl\text{-}proj}_1(A, a, c), \mathsf{refl\text{-}proj}_2(A, a, c)) = c : \mathbf{Id}(A, a, a).$$

But this is tantamount to

$$\mathsf{refl}(A, a) = c : \mathbf{Id}(A, a, a). \qquad (\mathbf{Id}\text{-}\eta)$$

§4. **Nonderivability of eta rules.** It is intuitively clear that when the c appearing in the formulation of the eta rules above is a variable x, then the resulting judgement is not derivable in Martin-Löf type theory. There simply are no rules that allow one to demonstrate, for instance,

$$x : \Sigma(A, B) \vdash \mathsf{pair}(\mathsf{fst}(x), \mathsf{snd}(x)) = x : \Sigma(A, B).$$

The nonderivability of eta rules can be established more rigorously by reference to the normalization theorem for Martin-Löf type theory. A corollary of this theorem is the Church–Rosser theorem, namely, that if

$$a = b : A$$

is derivable, then there is a $c : A$ such that both a and b reduce to c. Here the reduction relation is an asymmetric version of the relation of judgemental identity. A variable x reduces only to itself, as does $\mathsf{pair}(\mathsf{fst}(x), \mathsf{snd}(x))$, since $\mathsf{pair}(a, b)$ is reducible if and only if either a or b is reducible, but neither $\mathsf{fst}(x)$ nor $\mathsf{snd}(x)$ is reducible.

Of the normalization proofs available in the literature, the one developed by Goguen [10] seems especially pertinent to our context. The system studied by Goguen is type theory in its higher order formulation with a scheme in the style of Dybjer for defining sets and their constructors and selectors. Goguen's proof, moreover, covers the normalization of open terms, on which we have relied above; and the Church–Rosser theorem is explicitly stated as a corollary (ibid. Corollary 6.8.6).

§5. Propositional eta identities. Whenever $a, b : A$, one may form the identity proposition $\mathbf{Id}(A, a, b)$. Let us call it the identity proposition corresponding to the identity judgement $a = b : A$. An identity proposition corresponding to a (judgemental) eta identity may be called an eta proposition. The following, for instance, is an eta proposition:

$$\mathbf{Id}(\Sigma(A, B), \mathsf{pair}(\mathsf{fst}(c), \mathsf{snd}(c)), c).$$

Let us call a judgement to the effect that an eta proposition is inhabited a propositional eta rule. Whereas the ordinary (judgemental) eta rules are not derivable in Martin-Löf type theory, propositional eta rules, in many cases, are derivable. The required proof object is, in general, constructed by a single application of the corresponding elimination rule. Thus, for Σ we have

$$\mathsf{E}\big(c, [x][y]\mathsf{refl}(\mathsf{pair}(x, y))\big) : \mathbf{Id}(\Sigma(A, B), \mathsf{pair}(\mathsf{fst}(c), \mathsf{snd}(c)), c).$$

And for Π we have

$$\mathsf{F}(c, [f]\mathsf{refl}(\lambda(f)) : \mathbf{Id}(\Pi(A, B), \lambda([x]\mathsf{ap}(c, x)), c).$$

Here we are relying on the selector F available in the higher order formulation. Garner [9] shows that the propositional eta rule for Π is not derivable in the lower order formulation.

The propositional eta rules for $W(A, B)$ and $\mathbf{1}$ can be demonstrated similarly.

Before considering the propositional eta rule for $\mathbf{Id}(A, a, a)$, let us note that any set A allows the formation of a proposition saying that an arbitrary element $c : A$ is propositionally identical to an element of constructor form. Here are three examples of such generalized eta propositions:

$$\begin{array}{l} \mathbf{Id}(\mathbf{2}, \mathsf{t}, c) \vee \mathbf{Id}(\mathbf{2}, \mathsf{f}, c), \\ \mathbf{Id}(\mathbf{N}, 0, c) \vee (\exists x : \mathbf{N})\mathbf{Id}(\mathbf{N}, \mathsf{s}(x), c), \\ (\exists x : A)\mathbf{Id}(A + B, \mathsf{inl}(x), c) \vee (\exists y : B)\mathbf{Id}(A + B, \mathsf{inr}(y), c). \end{array}$$

Here $\mathbf{N}$ is the set of natural numbers and has constructors $0 : \mathbf{N}$ and $\mathsf{s} : (\mathbf{N})\mathbf{N}$. The disjoint union $A + B : \mathit{set}$ has the two constructors

$$\begin{array}{l} \mathsf{inl} : (A)A + B, \\ \mathsf{inr} : (B)A + B. \end{array} \qquad (+\text{-intro})$$

The propositional eta rule for $\mathbf{Id}$ is not in general derivable in the type theory assumed here. The eta proposition in question is

$$\mathbf{Id}(\mathbf{Id}(A, a, a), \mathsf{refl}(a), c) \qquad (\mathbf{Id}\text{-}\eta\text{-prop-}A)$$

for arbitrary $a : A$ and $c : \mathbf{Id}(A, a, a)$. That this set is in general not inhabited was shown by Hofmann and Streicher [15]. But although we do not have inhabitation for arbitrary A, there are sets A for which ($\mathbf{Id}$-η-prop-A) is inhabited. Hofmann [14, p. 58] showed it to be inhabited for $\mathbf{0}$, $\mathbf{1}$, and $\mathbf{N}$. Hedberg [13] showed more generally that ($\mathbf{Id}$-η-prop-A) is inhabited whenever the relation $\mathbf{Id}(A, x, y)$ is decidable, meaning that there is a function $d : (x, y : A)\, \mathbf{Id}(A, x, y) + \neg\mathbf{Id}(A, x, y)$. That the identity relations on $\mathbf{0}$ and $\mathbf{1}$ are decidable is obvious; that $\mathbf{N}$ has a decidable identity relation can be shown by a double induction.

§6. Eta identities in category theory. The rules (Σ-η), (Π-η), and ($\mathbf{1}$-η) have counterparts in the theory of cartesian closed categories.

Let us first briefly recall the notion of a category (we follow the presentation of [16]). There is a collection of objects $A, B, C, \ldots$; a collection of arrows $f, g, h, \ldots$; and two functions, source and target, from the arrows into the objects. The collection of objects and the collection of arrows are both equipped with identity. One writes $f : A \to B$ to mean source$(f) = A$ and target$(f) = B$. Whenever $f : A \to B$ and $g : B \to C$, there is a composite arrow $gf : A \to C$. Arrow composition is associative, thus $(hg)f = h(gf)$. With every object A there is associated an arrow 1_A satisfying $f1_A = f : A \to B$ and $1_A g = g : B \to A$.

A category is cartesian closed if it has the following additional structure.

For any objects A, B there is an object $A \times B$, called a product of A and B; two arrows $\pi_{A,B} : A \times B \to A$ and $\pi'_{A,B} : A \times B \to B$; and whenever $f : C \to A$ and $g : C \to B$, then there is $\langle f, g\rangle : C \to A \times B$ such that the following equations hold:

$$\pi_{A,B}\langle f, g\rangle = f : C \to A,$$
$$\pi'_{A,B}\langle f, g\rangle = g : C \to B.$$

Moreover, for any $h : C \to A \times B$ we require

$$\langle \pi_{A,B}h, \pi'_{A,B}h\rangle = h : C \to A \times B. \qquad \text{(prod-}\eta\text{)}$$

For any objects A, B there is an object B^A, called an exponential; an arrow $\varepsilon_{A,B} : B^A \times A \to B$, called evaluation; and whenever $f : C \times A \to B$, then there is an arrow $f^* : C \to B^A$ such that the following equation holds:

$$\varepsilon_{A,B}\langle f^*\pi_{C,A}\, ,\, \pi'_{C,A}\rangle = f : C \times A \to B.$$

If we regard $f : C \times A \to B$ as a term of type B depending on the types C and A, then we may regard f^* as the λ-abstraction of f with respect to type A. For any $h : C \to B^A$ we require

$$(\varepsilon_{A,B}\langle h\pi_{C,A}\, ,\, \pi'_{C,A}\rangle)^* = h : C \to B^A. \qquad \text{(exp-}\eta\text{)}$$

There is an object $\mathbf{1}$, called a terminal object; and for any object A an arrow $\bigcirc_A : A \to \mathbf{1}$. For any $f : A \to \mathbf{1}$ we require

$$\bigcirc_A = f : A \to \mathbf{1}. \qquad \text{(term-}\eta\text{)}$$

The equation (prod-η) secures that any arrow $h : C \to A \times B$ such that

$$\pi_{A,B}h = f : C \to A,$$
$$\pi'_{A,B}h = g : C \to B$$

is equal to $\langle f, g\rangle$. Namely, $h = \langle \pi_{A,B}h, \pi'_{A,B}h\rangle = \langle f, g\rangle$. Likewise, the equation (exp-η) secures that any arrow $h : C \to B^A$ such that

$$\varepsilon_{A,B}\langle h\pi_{C,A}, \pi'_{C,A}\rangle = f : C \times A \to B$$

is equal to f^*. Namely, $h = (\varepsilon_{A,B}\langle h\pi_{C,A}, \pi'_{C,A}\rangle)^* = f^*$. And the equation (term-η) secures that any $f : A \to \mathbf{1}$ is equal to $\bigcirc_A$.

The equations (prod-η), (exp-η), and (term-η) may therefore be called uniqueness principles, since they entail the uniqueness of certain arrows associated with the category-theoretic constructions of product, exponential, and terminal object. The uniqueness of these arrows is needed, for instance, in showing that each of these constructions is unique up to isomorphism. When the uniqueness principles are dispensed with, the resulting constructions are called weak (see, e.g., [2]).

A so-called bicartesian closed category is additionally equipped with the following structure.

For any objects A, B there is an object $A + B$, called a co-product of A and B; two arrows $\iota_{A,B} : A \to A + B$ and $\iota'_{A,B} : B \to A + B$; and whenever $f : A \to C$ and $g : B \to C$, then there is $[f, g] : A + B \to C$ such that the following equations hold:

$$\begin{aligned}[f, g]\,\iota_{A,B} &= f : A \to C,\\ [f, g]\,\iota'_{A,B} &= g : B \to C.\end{aligned}$$

Moreover, for any $h : A + B \to C$ we require

$$[h\iota_{A,B}, h\iota'_{A,B}] = h : A + B \to C. \qquad \text{(co-prod-}\eta)$$

There is an object $\mathbf{0}$, called an initial object; and for any object A an arrow $\Box_A : \mathbf{0} \to A$. For any $f : \mathbf{0} \to A$ we require

$$\Box_A = f : \mathbf{0} \to A. \qquad \text{(init-}\eta)$$

Whereas the uniqueness principles (prod-η), (exp-η), and (term-η) concern arrows whose *target* is a product, an exponential, or a terminal object, the uniqueness principles (co-prod-η) and (init-η) concern arrows whose *source* is a co-product or an initial object. This dual form of uniqueness principle suggests the formulation of what we shall call co-eta rules.

§7. Co-eta rules.

An eta rule for a set A asserts that any element of A is judgementally identical to an element of constructor form. A co-eta rule for A asserts that any function $f : (x : A)C(x)$ applied to an arbitrary element $c : A$ is judgementally identical to an element of selector form.

The disjoint union $A + B : set$ has a selector D satisfying

$$\begin{aligned}\mathsf{D}(\mathsf{inl}(a), f, g) &= f(a) : C(\mathsf{inl}(a)),\\ \mathsf{D}(\mathsf{inr}(b), f, g) &= g(b) : C(\mathsf{inr}(b)),\end{aligned} \qquad (+\text{-eq})$$

where $C : (A + B)set$, $f : (x : A)C(\mathsf{inl}(x))$, and $g : (y : B)C(\mathsf{inr}(y))$. Let

$$h : (z : A + B)C(z).$$

The co-eta rule for disjoint union is then:

$$\mathsf{D}\big(c, [x]h(\mathsf{inl}(x)), [y]h(\mathsf{inr}(y))\big) = h(c) : C(c). \qquad (+\text{-co-}\eta)$$

This rule says that h applied to an arbitrary element $c : A + B$ is judgementally identical to an element of selector form. The following rule of inductive extensionality is, as we shall see, equivalent:

$$\frac{c : A + B \qquad \begin{array}{l} x : A \vdash h(\mathsf{inl}(x)) = h'(\mathsf{inl}(x)) : C(\mathsf{inl}(x)) \\ y : B \vdash h(\mathsf{inr}(y)) = h'(\mathsf{inr}(y)) : C(\mathsf{inr}(y)) \end{array}}{h(c) = h'(c) : C(c).} \qquad (+\text{-ind-ext})$$

The second and third premiss are here written in one column for reasons of space. This rule says that if h and h' are judgementally identical on any argument of canonical form, then they are judgementally identical on any element $c : A + B$. In the presence of ordinary function extensionality, (Ext), this entails that h and h' themselves are judgementally identical, $h = h' : (z : A + B)C(z)$.

One sees the equivalence of ($+$-co-η) and ($+$-ind-ext) as follows. Assume the premisses of ($+$-ind-ext) to be known. Then, by (ξ) and (App-cong), we may infer:

$$\mathsf{D}\big(c, [x]h(\mathsf{inl}(x)), [y]h(\mathsf{inr}(y))\big) = \mathsf{D}\big(c, [x]h'(\mathsf{inl}(x)), [y]h'(\mathsf{inr}(y))\big) : C(c).$$

From ($+$-co-η) we thence get

$$h(c) = h'(c) : C(c),$$

the conclusion of ($+$-ind-ext). On the other hand, from ($+$-eq) we get

$$\begin{aligned} x : A &\vdash \mathsf{D}\big(\mathsf{inl}(x), [x]h(\mathsf{inl}(x)), [y]h(\mathsf{inr}(y))\big) = h(\mathsf{inl}(x)) : C(\mathsf{inl}(x)), \\ y : B &\vdash \mathsf{D}\big(\mathsf{inr}(y), [x]h(\mathsf{inl}(x)), [y]h(\mathsf{inr}(y))\big) = h(\mathsf{inr}(y)) : C(\mathsf{inr}(y)), \end{aligned}$$

whence ($+$-ind-ext) allows us to infer

$$\mathsf{D}\big(c, [x]h(\mathsf{inl}(x)), [y]h(\mathsf{inr}(y))\big) = h(c) : C(c).$$

A similar equivalence holds for co-eta rules in general. In some formal languages, such as in the type theory as formulated in [20], there are no selectors, but instead schemes for defining functions by induction. In such a language, of course, we cannot formulate co-eta rules, but rules of inductive extensionality are formulable.

The set $\Sigma(A, B)$ has a selector E satisfying

$$\mathsf{E}(\mathsf{pair}(a, b), f) = f(a, b) : C(\mathsf{pair}(a, b)), \qquad (\Sigma\text{-eq})$$

where $C : (\Sigma(A, B))\mathit{set}$ and $f : (x : A)(y : B(x))\, C(\mathsf{pair}(x, y))$. For any

$$h : (z : \Sigma(A, B))\, C(z),$$

we have

$$[x, y]h(\mathsf{pair}(x, y)) : (x : A)(y : B(x))\, C(\mathsf{pair}(x, y)).$$

Thus a co-eta rule may be formulated as follows:

$$\mathsf{E}\big(c, [x, y]h(\mathsf{pair}(x, y))\big) = h(c) : C(c). \qquad (\Sigma\text{-co-}\eta)$$

Since in the higher order formulation of Martin-Löf type theory each set has a unique selector, one can there formulate a co-eta rule for any set. Relying on the Dybjer scheme for elimination rules, we may describe a procedure for formulating the co-eta rule for A. The selector of A has among its argument places one for each constructor of A. For instance, the selector E has one such argument place, corresponding to the constructor pair; and the selector D has two such argument places, one corresponding to the constructor inl and one to the constructor inr. Suppose we are given $h : (x : A)C(x)$. For each constructor con of A, apply h to a term of the form $\mathsf{con}(x_1, \ldots, x_n)$. If con is a constant, then the sequence of variables here is empty. But if con is a function, then $h(\mathsf{con}(x_1, \dots, x_n))$ is an open term that cannot be inserted directly into the appropriate argument place of the selector. We obtain a closed term by means of abstraction. To obtain a term of the right type it is, however, not in general enough to abstract only the variables $x_1, \dots, x_n$, since the syntax of selectors allows also for recursive arguments. (An illustration of this will be given shortly.) Insert each closed term thus obtained through abstraction at the corresponding argument place in the selector. The equality rules for A secures that the resulting term is judgementally identical to h on any argument $a : A$ of canonical form. The co-eta rule asserts that the terms are judgementally identical on an arbitrary argument.

It is easy to see that the co-eta rules given above for $A + B$ and $\Sigma(A, B)$ follow the described pattern. For an illustration of a selector with a recursive argument, let us consider the set of natural numbers, **N**. For any $C : (\mathbf{N})set$, $b : C(0)$ and $f : (x : N)(y : C(x))\, C(\mathsf{s}(x))$, its selector R satisfies

$$\begin{aligned} &\mathsf{R}(0, b, f) = b : C(0), \\ &\mathsf{R}(\mathsf{s}(n), b, f) = f(n, \mathsf{R}(n, b, f)) : C(\mathsf{s}(n)). \end{aligned} \tag{$\mathbf{N}$-eq}$$

The third argument of R—associated with the constructor s and here written f—is recursive. Given a function $h : (x : \mathbf{N})C(x)$, we apply it to the constructor 0, obtaining $h(0) : C(0)$; and to an arbitrary element of s-form, obtaining $h(\mathsf{s}(x)) : C(\mathsf{s}(x))$ for $x : \mathbf{N}$. In order to get a term of appropriate type from $h(\mathsf{s}(x))$ we must abstract not only x but also $y : C(x)$, as required by the recursive argument f in $\mathsf{R}(n, b, f)$. We thus get $[x : \mathbf{N}][y : C(x)]h(\mathsf{s}(x))$, and so we may formulate

$$\mathsf{R}\big(n, h(0), [x : \mathbf{N}][y : C(x)]h(\mathsf{s}(x))\big) = h(n) : C(n). \tag{$\mathbf{N}$-co-η}$$

The equivalent rule of inductive extensionality is[2]

$$\frac{\begin{array}{cc} & h(0) = h'(0) : C(0) \\ n : \mathbf{N} & x : \mathbf{N} \vdash h(\mathsf{s}(x)) = h'(\mathsf{s}(x)) : C(\mathsf{s}(x)) \end{array}}{h(n) = h'(n) : C(n).} \tag{$\mathbf{N}$-ind-ext}$$

The co-eta rule for identity sets illustrates a further generalization. Let us first recall the rule

$$\frac{p : \mathbf{Id}(A, a, b) \qquad d : (z : A)\, C(z, z, \mathsf{refl}(z))}{\mathsf{J}(a, b, p, d) : C(a, b, p),} \tag{$\mathbf{Id}$-elim}$$

where $C : (x : A)(y : A)(\mathbf{Id}(A, x, y))\, set$. Assume

$$h : (x : A)(y : A)(p : \mathbf{Id}(A, x, y))\, C(x, y, p).$$

[2] Another extensionality principle on **N** is the rule of uniqueness of definition by induction:

$$\frac{\begin{array}{ccc} & & x : \mathbf{N} \vdash h(\mathsf{s}(x)) = f\big(x, h(x)\big) : C(\mathsf{s}(x)) \\ n : \mathbf{N} & h(0) = h'(0) : C(0) & x : \mathbf{N} \vdash h'(\mathsf{s}(x)) = f\big(x, h'(x))\big) : C(\mathsf{s}(x)) \end{array}}{h(n) = h'(n) : C(n).}$$

This rule, studied for instance in [11] and [25], entails (**N**-ind-ext), and would indeed seem to be stronger than it.

Thus we assume given a function on the inductive family $(x : A)(y : A)\,\mathbf{Id}(A, x, y)$ rather than on the set $\mathbf{Id}(A, a, a)$. We may then formulate a co-eta rule as follows:

$$\mathsf{J}\big(a, b, p, [z]h(z, z, \mathsf{refl}(A, z))\big) = h(a, b, p) : C(a, b, p). \qquad (\mathbf{Id}\text{-co-}\eta)$$

This identity has been studied by Streicher [31, pp. 15–18], who shows that it entails the so-called reflection rule:

$$\frac{p : \mathbf{Id}(A, a, b)}{a = b : A.}$$

Namely, let

$$x, y : A, p : \mathbf{Id}(A, x, y) \vdash h_1(x, y, p) = x : A,$$
$$x, y : A, p : \mathbf{Id}(A, x, y) \vdash h_2(x, y, p) = y : A.$$

Then

$$z : A \vdash h_1(z, z, \mathsf{refl}(A, z)) = h_2(z, z, \mathsf{refl}(A, z)) : A, \qquad (1)$$

whence, by (ξ) and (App-cong),

$$\mathsf{J}\big(x, y, p, [z]h_1(z, z, \mathsf{refl}(A, z))\big) = \mathsf{J}\big(x, y, p, [z]h_2(z, z, \mathsf{refl}(A, z))\big) : A.$$

By $(\mathbf{Id}\text{-co-}\eta)$ this yields

$$x, y : A, p : \mathbf{Id}(A, x, y) \vdash h_1(x, y, p) = h_2(x, y, p) : A. \qquad (2)$$

In light of the definitions of h_1 and h_2, this allows us to infer $a = b : A$ from $p : \mathbf{Id}(A, a, b)$. The step from (1) to (2) can be seen as an inference by inductive extensionality: if h_1 and h_2 agree on canonical triples $(z, z, \mathsf{refl}(z))$, for $z : A$, then they agree on arbitrary triples (x, y, p), where $p : \mathbf{Id}(A, x, y)$.

In a similar fashion one can show that $(\mathbf{Id}\text{-co-}\eta)$ entails $(\mathbf{Id}\text{-}\eta)$. Namely, let

$$x, y : A, p : \mathbf{Id}(A, x, y) \vdash g_1(x, y, p) = p : \mathbf{Id}(A, x, y),$$
$$x, y : A, p : \mathbf{Id}(A, x, y) \vdash g_2(x, y, p) = \mathsf{refl}(A, x) : \mathbf{Id}(A, x, y).$$

Assuming $(\mathbf{Id}\text{-co-}\eta)$, we can avail ourselves of the reflection rule, hence in the context $x, y : A, p : \mathbf{Id}(A, x, y)$ we have $\mathbf{Id}(A, x, y) = \mathbf{Id}(A, x, x) : \mathit{set}$, whence the function g_2 is well-typed. As before we can use $(\mathbf{Id}\text{-co-}\eta)$ to conclude

$$\mathsf{refl}(A, a) = c : \mathbf{Id}(A, a, a)$$

for arbitrary $a : A$ and $c : \mathbf{Id}(A, a, a)$.

It is easy to see from the description of the general form of co-eta rules that the corresponding propositional co-eta rules are derivable. (This was noted already by Luo [18, p. 201], who calls co-eta propositions 'filling-up rules.') Namely, the co-eta proposition for A is inhabited for canonical elements of A by the equality rule(s) for A and $\mathbf{Id}$-introduction; by the elimination rule for A it then follows that its co-eta proposition is inhabited for arbitrary elements of A. This argument generalizes directly to the case where A is an inductive family rather than an inductive set. In particular, it is easy to see that any proposition of the form

$$\mathbf{Id}\big(C(a, b, p), \mathsf{J}\big(a, b, p, [z]h(z, z, \mathsf{refl}(z))\big), h(a, b, p)\big)$$

is inhabited for $a, b : A$ and $p : \mathbf{Id}(A, a, b)$. It follows that the reflection rule and $(\mathbf{Id}\text{-co-}\eta)$ are interderivable, as already noted by Streicher.

§8. Eta identities in proof theory. Prawitz [27, p. 254] briefly discusses the notion of an expansion of a natural deduction derivation. When the main connective of the end formula of a derivation $\mathcal{D}$ is $\wedge$, $\supset$, or $\vee$, then the immediate expansion of $\mathcal{D}$ is formed as follows:

$$\begin{array}{c}\mathcal{D}\\ A\wedge B\end{array} \quad\rightsquigarrow\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A\wedge B\end{array}}{A} \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A\wedge B\end{array}}{B}}{A\wedge B} \tag{$\wedge$-exp}$$

$$\begin{array}{c}\mathcal{D}\\ A\supset B\end{array} \quad\rightsquigarrow\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A\supset B\end{array} \qquad [A]}{B}}{A\supset B} \tag{$\supset$-exp}$$

$$\begin{array}{c}\mathcal{D}\\ A\vee B\end{array} \quad\rightsquigarrow\quad \dfrac{\begin{array}{c}\mathcal{D}\\ A\vee B\end{array} \qquad \dfrac{[A]}{A\vee B} \qquad \dfrac{[B]}{A\vee B}}{A\vee B.} \tag{$\vee$-exp}$$

Immediate expansions can be formed similarly for quantified formulae. These expansions were considered by Prawitz in the course of a discussion of the form of normal natural deduction derivations. In a proof of normalization it is of course reductions that play the main role, that is, operations on natural deduction derivations such as the following:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ A\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ B\end{array}}{A\wedge B}}{A} \quad\rightsquigarrow\quad \begin{array}{c}\mathcal{D}_1\\ A\end{array} \tag{$\wedge$-red$_1$}$$

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_1\\ B\end{array}}{A\supset B} \qquad \begin{array}{c}\mathcal{D}_2\\ A\end{array}}{B} \quad\rightsquigarrow\quad \begin{array}{c}\mathcal{D}_2\\ A\\ \mathcal{D}_1\\ B\end{array}. \tag{$\supset$-red}$$

But by also making use of expansions, a derivation of a formula A can be transformed into a normal derivation of A in which all the so-called minimum formulae are atomic. For instance, the normal derivation

$$\dfrac{\dfrac{A\wedge(B\wedge C)}{B\wedge C}}{A\wedge(B\wedge C)\supset(B\wedge C)}$$

has one minimum formula, $B\wedge C$, but it is composite. Employing ($\wedge$-exp) one obtains the normal derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{A\wedge(B\wedge C)}{B\wedge C}}{B} \qquad \dfrac{\dfrac{A\wedge(B\wedge C)}{B\wedge C}}{C}}{B\wedge C}}{A\wedge(B\wedge C)\supset(B\wedge C)}$$

in which there are two minimum formulae, B and C, of smaller complexity.

Immediate expansions are also of relevance to the question of the general form of introduction and elimination rules. (In type theory this form is described by the Dybjer schemes.) Pfenning and Davies [26] call the elimination rules for a connective Φ locally sound if Φ admits of a suitable number of reductions; and locally complete if Φ admits of an immediate expansion. Local soundness ensures that the elimination rules for Φ are not too strong, whereas local completeness ensures that its elimination rules are strong enough, in particular strong enough to recover the premisses of its introduction rules. Thus, if Φ is locally complete, then any formula whose main connective is Φ can be derived from itself by use of the introduction and elimination rules for Φ, as can be seen above for the case of $\wedge$, $\supset$, and $\vee$.

In type-theoretical notation, the right hand side of ($\vee$-exp) is written as $\mathsf{D}(c, [x]\mathsf{inl}(x), [y]\mathsf{inr}(y))$ rather than as $\mathsf{D}\big(c, [x]h(\mathsf{inl}(x)), [y]h(\mathsf{inr}(y))\big)$, which is the term occurring in (+-co-η). The expansion whose right hand side corresponds to this latter term is rather

$$\begin{array}{c}\mathcal{D}_1\\ A\vee B\\ \mathcal{D}_2\\ C\end{array} \quad\rightsquigarrow\quad \dfrac{\begin{array}{c}\mathcal{D}_1\\ A\vee B\end{array} \qquad \begin{array}{c}\dfrac{[A]}{A\vee B}\\ \mathcal{D}_2\\ C\end{array} \qquad \begin{array}{c}\dfrac{[B]}{A\vee B}\\ \mathcal{D}_2\\ C\end{array}}{C}\,.$$

The expansion ($\vee$-exp) is the special case of this where $\mathcal{D}_2$ is empty and C is $A \vee B$. Expansions of this more general kind have recently been studied by Tranchini [35]. He shows that in Schroeder-Heister's formalism of natural deduction with higher order rules [28], such a generalized expansion can be formulated for any connective. In type-theoretical notation, this general form of expansion is in effect just the general form of co-eta rules described above restricted to selectors without recursive arguments.

Prawitz [27, p. 257] formulated the conjecture, which he (ibid. p. 261) attributed to Martin-Löf, that identity between natural deduction derivations is the equivalence relation generated by reductions. The conjecture as stated does thus not say that a derivation $\mathcal{D}$ and its expansion are always identical, but Prawitz (ibid. p. 257) remarks that 'it seems unlikely that any interesting property of proofs is sensitive to differences created by an expansion'. Martin-Löf [19, p. 104] noted that the meaning of the conjecture depends on whether by identity between derivations one has in mind definitional identity, $\mathcal{D} \equiv \mathcal{D}'$—more on which below—or rather the notion of identity captured by the identity proposition $\mathbf{Id}(C, \mathcal{D}, \mathcal{D}')$. Indeed, assuming the results of the present article, we may say that whereas expansion for most connectives (viz. save the identity predicate) preserves propositional identity, it does in general not preserve definitional identity.

In later literature on the identity of natural deduction derivations, such as [4] and [36], it appears to be taken for granted that such identity is preserved under expansion, although it is not specified then whether one has in mind definitional identity or some other notion of identity. In support of the view that expansion preserves identity a metamathematical result is often cited to the effect that the equivalence relation generated by reduction and expansion is the strongest candidate possible for an identity relation between proofs that does not trivialize it, that is, that does not entail the identification of all derivations of the same theorem (the result is restricted to the $\supset, \wedge$-fragment of propositional logic). The result knows many different proofs: type-theoretic [5, 30], category-theoretic [29], and proof-theoretic (though only for the $\supset$-fragment) [36].

§9. Eta rules and meaning explanations. Martin-Löf's meaning explanations for his type theory play an essential role in the justification of its rules. In particular, they support its so-called simple minded consistency: based on our understanding of the system as a meaningful symbolism we can see that it does not allow the derivation of a judgement of the form

$$a : \bot$$

where a is a closed term.

The meaning explanations can be said to provide the system with a basic semantics, basic in the sense that the terms in which they are formulated do not themselves call for explanation in yet other terms. The meaning explanations thus differ from model-theoretic semantics, where the symbols of a formal system are explained in terms of some other mathematical theory, typically set theory, whose primitive notions themselves are in need of explanation. The meaning explanations are therefore informal in the sense that they are not stated in terms of a mathematical theory. But their being informal does not mean that they are imprecise: on the contrary, they are quite precise and allow for a detailed justification of the rules of the system.

Readers of the book [22] may have noticed that the rules for Π offered there include

$$\lambda([x]\mathsf{ap}(c, x)) = c : \Pi(A, B). \tag{Π-η}$$

The rule is natural in the context of the so-called extensional version of type theory presented in that book. Indeed, (Π-η) together with the reflection rule for **Id** allows one to demonstrate the judgement

$$\mathsf{funext}(c, d) : (\forall x : A)\,\mathbf{Id}(B(x), \mathsf{ap}(c, x), \mathsf{ap}(d, x)) \supset \mathbf{Id}(\Pi(A, B), c, d),$$

saying that $c, d : \Pi(A, B)$ are propositionally identical provided they are pointwise propositionally identical.

In line with the general programme of the book [22], a justification of (Π-η) is offered on the basis of the meaning explanations. Any identity judgement $a = b : A$ has as presuppositions $a : A$ and $b : A$. In particular, (Π-η) has as a presupposition

$$c : \Pi(A, B).$$

Let us first assume that (Π-η) is made in the empty context. Then c is a closed term, so by the explanation of the form of judgement $a : A$ we know that c is a programme that, when evaluated, yields an element of $\Pi(A, B)$ of canonical form, namely, of the form $\lambda(f)$, where $f : (x : A)B(x)$. From the explanation of the form of judgement $a = b : A$, and that of a canonical element as a programme that evaluates to itself, we may infer

$$c = \lambda(f) : \Pi(A, B).$$

For any variable $x : A$, we then have

$$\mathsf{ap}(c, x) = \mathsf{ap}(\lambda(f), x) = f(x) : B(x).$$

The first identity follows from (App-cong), the second from Π-equality. By the rules (ξ) and (η) we may infer

$$[x]\mathsf{ap}(c, x) = [x]f(x) = f : (x : A)B(x)$$

and thence also

$$\lambda([x]\mathsf{ap}(c, x)) = \lambda(f) = c : \Pi(A, B).$$

Thus $(\Pi\text{-}\eta)$ has been justified in this case.

Assume next that $(\Pi\text{-}\eta)$ is made in a nonempty context Γ. Let $\overline{a}$ be an environment for Γ. That is, if Γ is $x_1 : A_1, \dots, x_n : A_n$, then $\overline{a}$ is a sequence $a_1, \dots, a_n$ such that

$$a_1 : A_1, \dots a_n : A_n[a_1, \dots, a_{n-1}].$$

Here postfixed square brackets indicate substitution. By the meaning explanation of the hypothetical judgement $\Gamma \vdash c : \Pi(A, B)$, we know

$$c[\overline{a}] : \Pi(A, B)[\overline{a}].$$

Since $c[\overline{a}]$ is a closed term and $\Pi(A, B)[\overline{a}]$ has the form $\Pi(A', B')$, where $A' : set$ and $B' : (A')set$, we may reason as above to find

$$\lambda([x]\mathsf{ap}(c[\overline{a}], x)) = \lambda(f) = c[\overline{a}] : \Pi(A, B)[\overline{a}].$$

By the meaning explanation of hypothetical judgements, this justifies

$$\Gamma \vdash \lambda([x]\mathsf{ap}(c, x)) = c : \Pi(A, B).$$

For the other eta rules, we shall consider only the categorical case. The general case follows just as in the justification of $(\Pi\text{-}\eta)$.

Let us first consider $(\Sigma\text{-}\eta)$. From $c : \Sigma(A, B)$ we know $c = \mathsf{pair}(a, b) : \Sigma(A, B)$ for some $a : A$, $b : B(a)$. We therefore have

$$\begin{aligned}\mathsf{pair}(\mathsf{fst}(c), \mathsf{snd}(c)) &= \mathsf{pair}\big(\mathsf{fst}(\mathsf{pair}(a, b)), \mathsf{snd}(\mathsf{pair}(a, b))\big)\\ &= \mathsf{pair}(a, b)\\ &= c : \Sigma(A, B).\end{aligned}$$

The justification of $(\mathbf{W}\text{-}\eta)$ is quite similar, relying on the properties of the two functions $\mathsf{sup\text{-}proj}_1$ and $\mathsf{sup\text{-}proj}_2$.

For the justification of $(\mathbf{1}\text{-}\eta)$, assume $c : \mathbf{1}$. Then c evaluates to an element of canonical form. But there is just one such element, namely, $0_{\mathbf{1}}$; hence $c = 0_{\mathbf{1}} : \mathbf{1}$.

A judgement of the form $c : \mathbf{Id}(A, a, b)$ is explained as follows: c is a programme that, when evaluated, yields an element of the form $\mathsf{refl}(D, d)$ where $D = A : set$, $d = a : A$, and $d = b : A$. In particular, the judgement $c : \mathbf{Id}(A, a, a)$ means that c evaluates to some $\mathsf{refl}(D, d)$ where $D = A : set$ and $d = a : A$. But then

$$\mathsf{refl}(A, a) = \mathsf{refl}(D, d) = c : \mathbf{Id}(A, a, a).$$

§10. Definitional identity. That eta rules are justified by the meaning explanations does, however, not force us to accept these rules as axioms of Martin-Löf type theory. The meaning explanations need not be the final arbiter in the question of which rules to accept. This is clear from the case of the so-called reflection rule:

$$\frac{p : \mathbf{Id}(A, a, b)}{a = b : A.}$$

The explanation of the form of judgement $p : \mathbf{Id}(A, a, b)$ can be seen to justify this rule. But the rule is not part of canonical Martin-Löf type theory. Indeed, several reasons can be cited for not accepting the reflection rule as the elimination rule for **Id**. The rule does not fit the pattern of introduction and elimination rules that the other set formers follow [31, p. 13]. Accepting the reflection rule as an axiom makes judgements of the forms $a : A$ and $a = b : A$ undecidable in the recursion-theoretic sense [14, pp. 62–63]. Since definitional identity should be decidable in this sense, it follows that the reflection rule does not accord with the understanding of judgemental identity as definitional identity. Finally, the equality rule that pairs with the reflection rule is $(\mathbf{Id}\text{-}\eta)$, and this judgement is inconsistent with the univalence axiom [34, Example 3.19].

There may, likewise, be several reasons for not accepting lower order eta rules. Here it will be argued that such rules do not accord with the understanding of judgemental identity as definitional identity.

Let us write $s \equiv t$ for 's is definitionally identical to t.' We say that judgements of the form $\alpha = \beta : type$ and $a = b : \alpha$ are sound for definitional identity if the following inferences are justified:

$$\frac{\alpha = \beta : type}{\alpha \equiv \beta} \qquad \frac{a = b : \alpha}{a \equiv b.}$$

A rule

$$\frac{J_1 \quad \dots \quad J_n}{\alpha = \beta : type} \qquad \frac{J_1 \quad \dots \quad J_n}{a = b : \alpha}$$

is sound for definitional identity if from the assumption that all premisses J_k of the relevant form are sound for definitional identity, we may infer that also the conclusion, $a = b : \alpha$ or $\alpha = \beta : type$, is sound.

The understanding of judgemental identity as definitional identity obviously requires that the rules of the system be sound for definitional identity. Most rules governing judgemental identity are stipulatory in nature and do not require justification in the way that, for instance, an elimination rule requires justification. In particular, we do not need to make the conclusion of such a rule evident on the assumption that we know the premisses. We must make sure that, if the conclusion of the rule is $a = b : \alpha$, say, then it is plain that both $a : \alpha$ and $b : \alpha$. But we are not obliged, on the basis of the meaning explanations and the assumption that the premisses of the rule are known, to make the judgement $a = b : \alpha$ evident, for we are simply stipulating that this judgement holds. If we are to understand judgemental identity as definitional identity, we are, however, obliged to justify that the rule is sound for definitional identity.

When carrying out such a justification we need to have a firm grasp of the notion of definitional identity. Our only way of attaining such a grasp, it would seem, is by reflection on the notion of definition and on definitional practice. Definitional identity is a relation between meaningful linguistic expressions, hence it is to be expected that a characterization of it will be sensitive to the underlying language. The characterization of definitional identity given by Curry and Feys [3, Section 2E] presupposes only the most basic ways of forming expressions. We say that definitional identity is an equivalence relation, $\equiv$, generated by all axioms of the form

$$\textit{definiendum} \equiv \textit{definiens}$$

and the following rule:

$$\frac{X \equiv Y \qquad Z \equiv Z'}{X \equiv Y'.} \tag{R}$$

Here Y' is the result of replacing an occurrence of Z in Y by Z'. The rule (R) formalizes the principle that definitional identity is preserved under substitution. In the context of combinatory logic, the rule is restricted to expressions built up solely by means of function application. We shall understand the preservation of definitional identity under substitution in the strongest possible sense: definitional identity licences substitution in all contexts. In particular, we shall take the rule (R) to apply also when the relevant occurrence of Z in Y is within the scope of a bound variable. The following will thus be valid applications of (R):

$$\frac{\int_0^2 2x + 2x\, dx \equiv \int_0^2 2x + 2x\, dx \qquad 2x + 2x \equiv 4x}{\int_0^2 2x + 2x\, dx \equiv \int_0^2 4x\, dx}$$

$$\frac{(\forall x : D)(P(x) \supset \bot) \equiv (\forall x : D)(P(x) \supset \bot) \qquad P(x) \supset \bot \equiv \neg P(x)}{(\forall x : D)\neg P(x)}.$$

Strong substitutability, as we shall call it, would seem to be justified by the understanding of definitional identity as identity of meaning. And it follows naturally from the view of a definition as a licence to substitute definiens for definiendum, and vice versa, in all contexts.

The rules of type conversion

$$\frac{a : \alpha \qquad \alpha = \beta : \mathit{type}}{a : \beta} \qquad\qquad \frac{a = b : \alpha \qquad \alpha = \beta : \mathit{type}}{a = b : \beta}$$

are clearly not instances of the rule (R). But they are justified by strong substitutability in the sense that we may infer, say, $a : \beta$ from $a : \alpha$ and $\alpha \equiv \beta$ by substituting β for the predicate α in $a : \alpha$. (The premiss $\alpha \equiv \beta$ is obtained from the assumption that $\alpha = \beta : \mathit{type}$ is sound for definitional identity.) For languages that accommodate typing judgements, $a : \alpha$ and $a = b : \alpha$, it is thus natural to postulate type conversion as a primitive rule of definitional identity.

§11. Soundness for definitional identity. Let us see how the two following rules may be shown, using the above characterization, to be sound for definitional identity:

$$\frac{f = g : (x : \alpha)\beta \qquad a = b : \alpha}{f(a) = g(b) : \beta[a/x]} \qquad \text{(App-cong)}$$

$$\frac{\alpha = \alpha' : \mathit{type} \qquad x : \alpha \vdash \beta = \beta' : \mathit{type}}{(x : \alpha)\beta = (x : \alpha')\beta' : \mathit{type}.} \qquad (\xi\text{-}\mathit{type})$$

In the case of (App-cong) it must be argued that if f and g, respectively a and b, are definitionally identical objects, then $f(a)$ and $g(b)$ are definitionally identical objects. The argument may be written out as follows:

$$\frac{\dfrac{f(a) \equiv f(a) \qquad a \equiv b}{f(a) \equiv f(b)} \qquad f \equiv g}{f(a) \equiv g(b).}$$

It should be emphasized that the individual steps here rely on the rule (R) and not on the rule (App-cong), whose soundness we are attempting to justify. In the analogous justification of (ξ-*type*) it is clear that the steps rely on (R):

$$\frac{\dfrac{(x : \alpha)\beta \equiv (x : \alpha)\beta \qquad \alpha \equiv \alpha'}{(x : \alpha)\beta \equiv (x : \alpha')\beta} \qquad \beta \equiv \beta'}{(x : \alpha)\beta \equiv (x : \alpha')\beta'.}$$

The justification of (ξ) is quite similar. In both cases we make use of strong substitutability, that is, substitutability in all contexts.[3]

From the foregoing one sees that definitional identity is a congruence relation with respect to the three basic operations of type theory, namely, function application, $f(a)$; function abstraction, $[x]b$; and type abstraction, $(x : \alpha)\beta$.

An equality rule is sound for definitional identity because it has the form of a definitional equation:

$$\mathit{definiendum} \equiv \mathit{definiens}.$$

In particular, the equality rules for the set former Φ serve as a definition of the selector sel associated with Φ. (We may also think of the elimination rule for Φ as being part of this definition, namely, as providing the typing information

[3]Martin-Löf in [19] denied that the ξ-rule of the λ-calculus is, in the present terminology, sound for definitional identity. I believe this can be explained as follows. When the applicative order of evaluation is assumed, as it is in Martin-Löf's earlier work, it is natural to require that if $X \equiv Y$, then X and Y evaluate to syntactically identical expressions [32]. Since it is difficult to make good sense of the evaluation of open expressions, in particular of expressions occurring within the scope of a bound variable, one then seems forced to say that $\lambda x.t \equiv \lambda x.t'$ only if t and t' are syntactically identical. This blocks the ξ-rule and therefore also strong substitutability. In [21] lazy evaluation is assumed instead of applicative order, and $X \equiv Y$ is taken to mean that X and Y evaluate to definitionally identical canonical objects. The ξ-rule now becomes the specification of how identical canonical objects of a Π-set are formed. Moreover, strong substitutability is no longer excluded, so the higher order rule (ξ) can be justified as above.

pertaining to sel.) It is a definition by induction on an inductively defined set. The definition may, but need not, be recursive in the sense of relying on 'previous' values of itself. The definition of the selector R given by ($\mathbf{N}$-eq) is recursive, but the definition of the selector D given by ($+$-eq) is not recursive.

As an example of a lower order eta rule, let us take

$$\mathsf{pair}(\mathsf{fst}(c), \mathsf{snd}(c)) = c : \Sigma(A, B). \tag{Σ-η}$$

Since the right-hand side of (Σ-η) is atomic, namely, a parameter c, it is clear that we cannot obtain it from the left-hand side by a sequence of applications of the rule (R), such as we obtained $g(b)$ from $f(a)$ in the justification of (App-cong) above. Therefore, the only way of regarding (Σ-η) as a definitional identity is by regarding it as a definition, namely, as being of the form

$$\textit{definiendum} \equiv \textit{definiens}.$$

But pair is a constructor and therefore a primitive function, hence it neither needs definition nor indeed admits of one. If we were to regard (Σ-η) as a definition, it would therefore have to be as a definition of the two projection functions fst and snd. But these have already been defined in terms of the selector E; and we have no right to define the same symbol twice.[4]

The arguments showing why (Π-η) and ($\mathbf{W}$-η) are not sound for definitional identity are quite similar. In the case of

$$0_{\mathbf{1}} = c : \mathbf{1} \tag{$\mathbf{1}$-η}$$

and

$$\mathsf{refl}(A, a) = c : \mathbf{Id}(A, a, a) \tag{$\mathbf{Id}$-η}$$

it is enough to notice that $0_{\mathbf{1}}$ and refl are constructors, whence they do not admit of definition. The displayed identities would moreover not seem to be felicitous as definitions, since the right-hand side—the purported definiens—includes a parameter, c, not present on the left-hand side.

Similar arguments work as well to show that co-eta rules are not sound for definitional identity. For instance, the identity

$$\mathsf{E}\big(c, [x, y]h(\mathsf{pair}(x, y))\big) = h(c) : C(c) \tag{Σ-co-η}$$

cannot be regarded as a definition of the selector E, which has already been defined by (Σ-eq). Nor is it possible to derive (Σ-co-η) from a definition by means of the rule (R).

Likewise, there is no way of deriving the equivalent rule of inductive extensionality from definitions by means of (R):

$$\frac{c : \Sigma(A, B) \qquad x : A, y : B(x) \vdash h(\mathsf{pair}(x, y)) = h'(\mathsf{pair}(x, y))}{h(c) = h'(c) : C(c).}$$

One may ask, however, whether inductive extensionality could be regarded as a primitive rule of definitional identity. The question is pertinent since

[4]The ban on defining the same symbol twice is Frege's second basic principle of definition in [8, Section 33]. It is implicit already in Aristotle's discussion of definition in *Topics* VI.4. This answers Tait's request [33, p. 170] for a principle that excludes (Σ-η) as a rule of definitional identity.

we shall argue below that ordinary extensionality, (Ext), is sound for definitional identity. As a principle of definitional identity, inductive extensionality would say: if h and h' are definitionally identical on all terms (closed or open) of constructor form, then they are definitionally identical on all terms. The only way of justifying this, however, would seem to be by saying that any term in the domain of h and h' is definitionally identical to a term of constructor form; but that is a lower order eta rule.

A principle similar to inductive extensionality deserves a brief discussion here. Uniqueness of definition by induction says that if the inductively defined functions h and h' have the same definition (in a suitable sense), then they are themselves definitionally identical. On $\Sigma(A, B)$ and $A + B$ this principle coincides with inductive extensionality, but on **N** and well-ordering sets, $\mathbf{W}(A, B)$, they come apart (see footnote 2). Definition by induction is of course a legitimate mode of definition; but there is a sense in which such a definition is incomplete, since the definiendum is required to have a special form. Thus, in a definition by induction on $\Sigma(A, B)$, the definiendum must have the form $h(\mathsf{pair}(x, y))$; and on **N** it must have the form $h(0)$ or $h(\mathsf{s}(x))$ (possibly with parameters). Suppose that for inductively defined functions $h, h' : (z : \mathbf{N})C(z)$, we have $h(0) \equiv h'(0)$ and

$$\begin{aligned} x : \mathbf{N} &\vdash h(\mathsf{s}(x)) \equiv f(x, h(x)), \\ x : \mathbf{N} &\vdash h'(\mathsf{s}(x)) \equiv f(x, h'(x)). \end{aligned}$$

Thus, h and h' have, in a suitable sense, the same definition. From the definitions of h and h' we may infer $h(z), h'(z) : C(z)$ for any variable $z : \mathbf{N}$. We cannot, however, infer anything as to what these $h(z)$ and $h'(z)$ are. All we can say is that they are arbitrary values of h and h' respectively. In particular, then, we cannot infer that $h(z)$ and $h'(z)$ are definitionally identical. The same argument applies, mutatis mutandis, to all other sets. Hence, uniqueness of definition by induction is not sound for definitional identity.

That the reflection rule is not sound for definitional identity has in effect already been noted. Indeed, it is difficult to see that we can infer that a and b are definitionally identical merely from the existence of a proof $p : \mathbf{Id}(A, a, b)$. If the reflection rule is assumed as the elimination rule for **Id**, the notions of judgemental identity and definitional identity thus come apart.

The inference

$$\frac{a : \neg A \qquad b : \neg B}{A = B : \mathit{set}}$$

appears to be justified by the meaning explanations. Assuming that we know the premisses, $a : \neg A$ and $b : \neg B$, we must make the judgement $A = B : \mathit{set}$ evident, that is, we must argue that any canonical element of A is a canonical element of B, and vice versa; and that identical canonical elements of A are identical canonical elements of B, and vice versa. Assume that c is a canonical element of A. Then $\mathsf{ap}(a, c) : \bot$. Since $\bot$ has no canonical elements, there can be no programme $\mathsf{ap}(a, c)$ that evaluates to a canonical element of $\bot$. Hence, by ex falso quodlibet, we may infer that c is also a canonical element of B. The rest of the argument is similar. The inference thus seems to be justified by the meaning explanations. But it is clearly not sound for definitional identity: we cannot infer that two sets A and B are definitionally identical simply on the grounds that neither is inhabited.

§12. Definitional identity at higher types. From a logico-grammatical point of view, the greatest difference between the lower order formulation and the higher order formulation of type theory is that in the latter, function symbols are categorematic, or self-standing. For instance, pair, pair(a), and pair(a, b) are there all well-formed terms, whereas in the lower order formulation only the last of these counts as well-formed.

In a higher order type theory with functional abstraction, one may therefore require that the explicit definition of a function symbol take the form

$$f \equiv t.$$

The explicit definition of a function symbol would thus be identical in form to the explicit definition of an individual constant symbol. It is, however, more in line with ordinary mathematical practice to allow the explicit definition of f to take the form

$$f(\overline{x}) \equiv t[\overline{x}]$$

where $t[\overline{x}]$ is some expression whose free variables are among $\overline{x}$.

That the rule

$$[x]f(x) = f : (x : \alpha)\beta \tag{η}$$

is sound for definitional identity can be seen by reflection on this common, and quite unobjectionable, practice of allowing the definiendum in an explicit definition of a function symbol f to take the form $f(\overline{x})$. For concreteness let us consider the following definition of a unary function $f : (\mathbf{N})\mathbf{N}$:

$$f(x) \equiv x^2 + x - 1. \tag{Def-f}$$

In the language of type theory, there seems to be only one way to justify viewing this as a definition of f, namely, by identifying f with the result of abstracting on the definiens of (Def-f). But this identification is justified only if f is definitionally identical to $[x]f(x)$. The following derivation spells out the reasoning.

$$\dfrac{\dfrac{[x]f(x) \equiv f}{f \equiv [x]f(x)} \qquad \dfrac{f(x) \equiv x^2 + x - 1}{[x]f(x) \equiv [x](x^2 + x - 1)}}{f \equiv [x](x^2 + x - 1).}$$

Thus it seems that mathematical practice, when seen through the lens of type theory, assumes (η) to be a primitive principle of definitional identity. For only such an assumption justifies regarding (Def-f) as constituting a definition of f.

Reflection on mathematical practice thus recommends that for a higher order language with functional abstraction, all well-typed identities of the form

$$[x]f(x) \equiv f$$

be taken as axioms of definitional identity. The higher order rule (η) is therefore sound for definitional identity.

This argument for the soundness of (η) is not in conflict with the argument against the soundness of $(\Pi\text{-}\eta)$. The soundness of

$$\lambda([x]\mathsf{ap}(c, x)) = c : \Pi(A, B) \tag{Π-η}$$

for definitional identity would require that it be regarded as a definition: not as a definition of λ, which is a constructor, but of ap. The function ap has, however, already been defined, either in terms of the selector F or—if ap is regarded as the selector—by the Π-equality rule. The rule (η), by contrast, is not postulated as a definition—of abstraction or function application, say—but as a primitive principle of definitional identity.

Nor is the argument for the soundness of (η) applicable to $(\Pi\text{-}\eta)$. The identity $(\Pi\text{-}\eta)$ cannot be regarded as a primitive principle of definitional identity, since it concerns the quite specific function symbols λ and ap and the quite specific type $\Pi(A, B)$. The rule (η), by contrast, concerns the general operations of abstraction and function application and the general function type, $(x : \alpha)\beta$.

For a Π-set we regard λ as primitive and ap as defined. At the function type $(x : \alpha)\beta$ the situation is in a sense the opposite: here application is primitive, whereas abstraction is defined. Namely, the function type $(x : \alpha)\beta$ is explained in terms of the two elimination-like rules (App) and (App-cong) governing the application operation. Abstraction $[x]b$ is then defined by the higher order rule

$$\frac{x : \alpha \vdash b : \beta \qquad a : \alpha}{([x]b)(a) = b[a/x] : \beta[a/x].} \tag{β}$$

In particular, the rule (β) is sound for definitional identity, since it serves as a definition of the abstraction operation. By stipulating the rule (Abs) we assert that $[x]b$ is a function of type $(x : \alpha)\beta$ whenever $x : \alpha \vdash b : \beta$. We explain which function it is through the rule (β), namely, by specifying its application behaviour. That (η) cannot be regarded as a definition of abstraction is clear, since the operand of the abstraction in $[x]f(x)$ has the special form $f(x)$ and not the general form b.

Since (ξ) and (η) are sound for definitional identity, it follows that also function extensionality is sound:

$$\frac{f, g : (x : \alpha)\beta \qquad x : \alpha \vdash f(x) = g(x) : \beta}{f = g : (x : \alpha)\beta.} \tag{Ext}$$

Consider for instance the following derivation:

$$\cfrac{\cfrac{\cfrac{[x]f(x) \equiv f}{f \equiv [x]f(x)} \qquad \cfrac{f(x) \equiv g(x)}{[x]f(x) \equiv [x]g(x)}}{f \equiv [x]g(x)} \qquad [x]g(x) \equiv g}{f \equiv g.}$$

An independent argument for the soundness of (Ext) can also be given. We want to justify the following rule:

$$\frac{f(x) \equiv g(x)}{f \equiv g.}$$

By the rules of substitution and the assumption that x is not free in f, we have, for any argument $a : \alpha$,

$$f(a) \equiv f(x)[a/x],$$

and likewise for g. Hence, if $f(x) \equiv g(x)$, then

$$f(a) \equiv f(x)[a/x] \equiv g(x)[a/x] \equiv g(a)$$

by strong substitutability. A function $f : (x : \alpha)\beta$ is determined by its application behaviour. Since f and g agree definitionally on each argument—they have definitionally identical application behaviour—they must themselves be definitionally identical. That is, we must have $f \equiv g$. Since (η) follows from (Ext) and (β), this gives an alternative argument for the soundness of (η).

§13. Acknowledgments. I am grateful to Per Martin-Löf for explaining to me his view on eta rules and to Luca Tranchini for discussion and comments. While writing the article I have been supported by grant no. 17-18344Y from the Czech Science Foundation, GAČR.

REFERENCES

[1] R. Backhouse, P. Chisholm, G. Malcolm, and E. Saaman, *Do-it-yourself type theory*. ***Formal Aspects of Computing***, vol. 1 (1989), pp. 19–84.

[2] S. Baranov and S. Soloviev, *Conditionally reversible computations and weak univerality in category theory*. ***Journal of Mathematical Sciences***, vol. 200 (2014), pp. 654–661.

[3] H. Curry and R. Feys, ***Combinatory Logic***, North-Holland, Amsterdam, 1958.

[4] K. Došen, *Identity of proofs based on normalization and generality*, this Bulletin, vol. 9 (2003), pp. 477–503.

[5] K. Došen and Z. Petrić, *The maximality of the typed lambda calculus and of cartesian closed categories*. ***Publications de l'Institut Mathématique***, vol. 68 (2000), pp. 1–19.

[6] P. Dybjer, *Inductive families*. ***Formal Aspects of Computing***, vol. 6 (1994), pp. 440–465.

[7] ———, *A general formulation of simultaneous inductive-recursive definitions in type theory*. ***The Journal of Symbolic Logic***, vol. 65 (2000), pp. 525–549.

[8] G. Frege, ***Grundgesetze der Arithmetik***, Hermann Pohle, Jena, 1893.

[9] R. Garner, *On the strength of dependent products in the type theory of Martin-Löf*. ***Annals of Pure and Applied Logic***, vol. 160 (2009), pp. 1–12.

[10] H. Goguen, ***A typed operational semantics for type theory***, Ph.D. thesis, University of Edinburgh, 1994.

[11] R. L. Goodstein, ***Recursive Number Theory***, North-Holland, Amsterdam, 1957.

[12] R. Harper, F. Honsell, and G. Plotkin, *A framework for defining logics*. ***Journal of the ACM***, vol. 40 (1993), pp. 143–184.

[13] M. Hedberg, *A coherence theorem for Martin-Löf's type theory*. ***Journal of Functional Programming***, vol. 8 (1998), pp. 413–436.

[14] M. Hofmann, ***Extensional Constructs in Intensional Type Theory***, Springer, London, 1997, Reprinted of Ph.D. thesis, University of Edinburgh, 1995.

[15] M. Hofmann and T. Streicher, *The groupoid interpretation of type theory*, ***Twenty-Five Years of Constructive Type Theory*** (G. Sambin and J. Smith, editors), Oxford University Press, Oxford, 1998, pp. 83–111.

[16] J. Lambek and P. J. Scott, ***Introduction to Higher Order Categorical Logic***, Cambridge University Press, Cambridge, 1986.

[17] P. Landin, *The mechanical evaluation of expressions*. ***The Computer Journal***, vol. 6 (1964), pp. 308–320.

[18] Z. Luo, ***Computation and Reasoning***, Clarendon Press, Oxford, 1994.

[19] P. Martin-Löf, *About models for intuitionistic type theories and the notion of definitional equality*, ***Proceedings of the Third Scandinavian Logic Symposium*** (S. Kanger, editor), North-Holland, Amsterdam, 1975, pp. 81–109.

[20] ———, *An intuitionistic theory of types: Predicative part*, ***Logic Colloquium '73*** (H. E. Rose and J. Shepherdson, editors), North-Holland, Amsterdam, 1975, pp. 73–118.

[21] ———, *Constructive mathematics and computer programming*, ***Logic, Methodology and Philosophy of Science, 1979*** (J. L. Cohen, J. Łoś, H. Pfeiffer, and K. P. Podewski, editors), North-Holland, Amsterdam, 1982, pp. 153–175.

[22] ———, ***Intuitionistic Type Theory***, Bibliopolis, Naples, 1984.

[23] B. Nordström, K. Petersson, and J. Smith, ***Programming in Martin-Löf's Type Theory***, Oxford University Press, Oxford, 1990.

[24] ———, *Martin-Löf's type theory*, ***Handbook of Logic in Computer Science. Volume 5: Logic and Algebraic Methods*** (S. Abramsky, D. Gabbay, and T. Maibaum, editors), Oxford University Press, Oxford, 2000, pp. 1–37.

[25] M. Okada and P. J. Scott, *A note on rewriting theory for uniqueness of iteration*. ***Theory and Applications of Categories***, vol. 6 (1999), pp. 47–64.

[26] F. Pfenning and R. Davies, *A judgemental reconstruction of modal logic*. ***Mathematical Structures in Computer Science***, vol. 11 (2001), pp. 511–540.

[27] D. Prawitz, *Ideas and results in proof theory*, ***Proceedings of the Second Scandinavian Logic Symposium*** (J. E. Fenstad, editor), North-Holland, Amsterdam, 1971, pp. 235–307.

[28] P. Schroeder-Heister, *A natural extension of natural deduction*. ***The Journal of Symbolic Logic***, vol. 49 (1984), pp. 1284–1300.

[29] A. Simpson, *Categorical completeness results for the simply-typed lambda-calculus*, ***Typed Lambda Calculi and Applications*** (M. Dezani-Ciancaglini, editor), Springer, Berlin, 1995, pp. 414–427.

[30] R. Statman, *λ-definable functionals and $\beta\eta$ conversion*. ***Archiv für mathematische Logik***, vol. 23 (1983), pp. 21–26.

[31] T. Streicher, ***Investigations into intensional type theory***, Habilitation thesis, Ludwig-Maximilian-University Munich, 1993.

[32] W. W. Tait, *Intensional interpretations of functionals of finite type I*. ***The Journal of Symbolic Logic***, vol. 32 (1967), pp. 198–212.

[33] ———, *Primitive recursive arithmetic and its role in the foundations of arithmetic: Historical and philosophical reflections*, ***Epistemology versus Ontology. Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf*** (P. Dybjer, S. Lindström, E. Palmgren, and G. Sundholm, editors), Springer, Dordrecht, 2012, pp. 161–180.

[34] The Univalent Foundations Program, ***Homotopy Type Theory: Univalent Foundations of Mathematics***, Institute for Advanced Study, Princeton, 2013. Available at `http://homotopytypetheory.org/book`.

[35] L. Tranchini, *Proof-theoretic harmony: Towards an intensional account*. ***Synthese*** (2016), DOI: `10.1007/s11229-016-1200-3`.

[36] F. Widebäck, ***Identity of Proofs***, Almqvist & Wiksell, Stockholm, 2001.

INSTITUTE OF PHILOSOPHY
CZECH ACADEMY OF SCIENCES
JILSKÁ 1, PRAGUE 1, 110 00, CZECH REPUBLIC
E-mail: klev@flu.cas.cz